

{ JSON }

JavaScript Object Notation

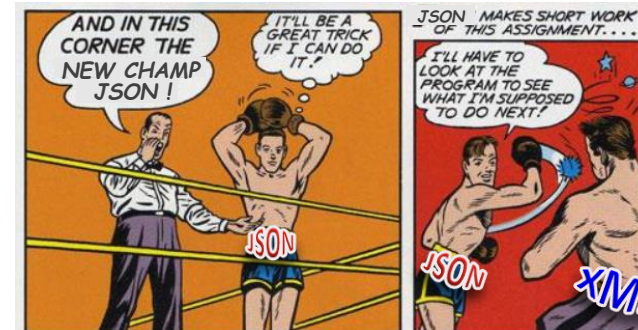


This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

JSON

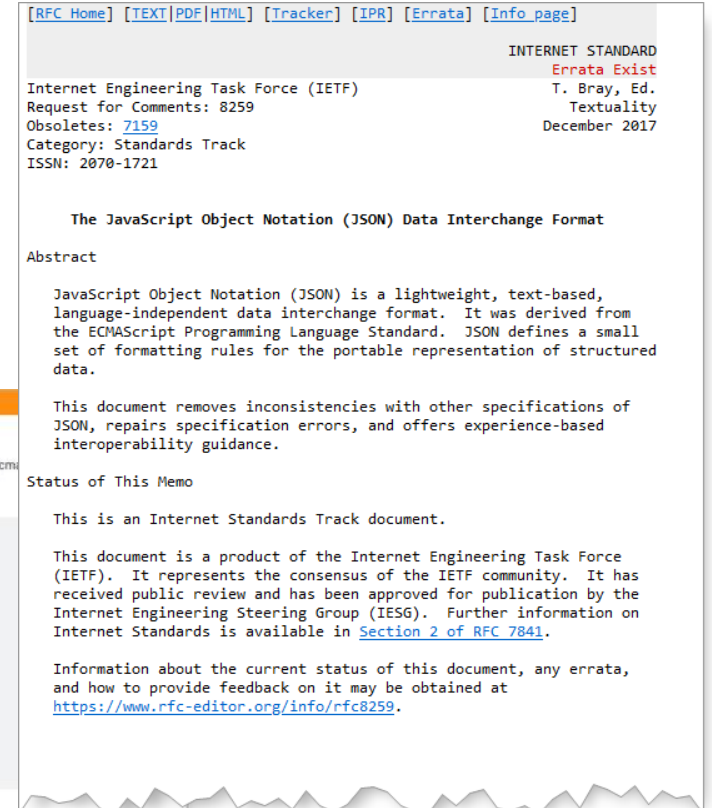
- JavaScript **O**bject **N**otation

- un format d'échange de données textuel "poids-léger" (lightweight) permettant de représenter des données structurées
 - plus léger que XML il est souvent plus rapide à lire et analyser (parse)
- indépendant de tout langage de programmation
 - bien que dérivé de la notation des objets dans le langage JavaScript
- auto-descriptif et facile à lire et comprendre pour un humain
- de plus en plus utilisé dans les API web où il a détrôné XML



JSON

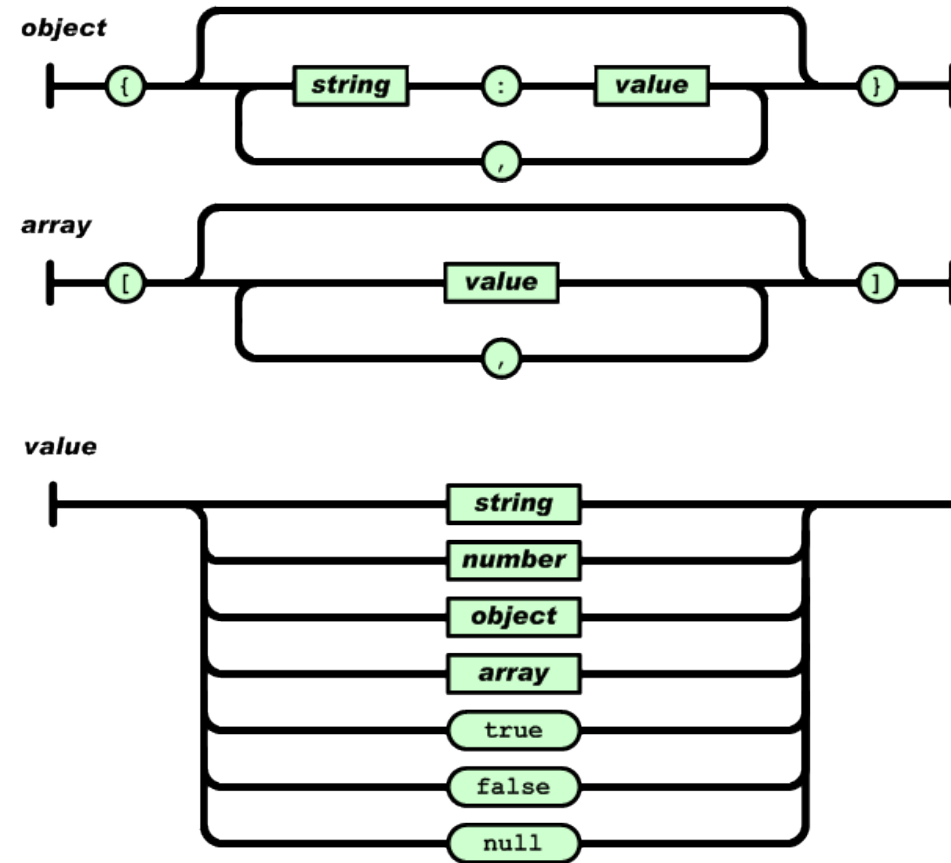
- Créé par Douglas Crockford entre 2002 et 2005
- est actuellement décrit par deux normes : [RFC 8259](#) de l'IETF et [ECMA-404](#) de l'ECMA
 - dernières versions décembre 2017



JSON is a lightweight, text-based, language-independent syntax for defining data interchange formats. It was derived from the ECMAScript programming language, but is programming language independent. JSON defines a small set of structuring rules for the portable representation of structured data.

JSON

- Basé sur deux structures universelles quasiment présentes dans tous les langages de programmation
 - **Collection de paires clé/valeur**
 - selon les langages object, record, struct, dictionnaire, tableau associatif...
 - **Liste ordonnées de valeurs**
 - selon les langages tableau, vecteur, liste ou séquence...
- Une valeur peut être :
 - une chaîne de caractères
 - un nombre
 - les valeurs `true`, `false` ou `null`
 - ou bien un objet ou tableau (ces structures peuvent être imbriquées)



figures issues de <http://www.json.org/>



JSON

Exemple

XML

un nom : Félicité
un âge : 10 ans
une race : chat de gouttière

chat

des activités préférées:
dormir,
manger du thon,
grimper aux arbres

JSON

```
<leschats>
  <chat nom = "Félicité">
    <age>10</age>
    <race>chat de gouttière</race>
    <aime>
      <item>manger du thon</item>
      <item>grimper aux arbres</item>
      <item>dormir</item>
    </aime>
    <poids>3.5</poids>
  </chat>
  <chat nom = "Felix">
    <age>6</age>
    <race>siamois</race>
    <aime>
      <item>se lécher</item>
      <item>manger des croquettes</item>
      <item>dormir</item>
    </aime>
    <poids>3.</poids>
  </chat>
</leschats>
```

moi j'aime
les croquettes



```
[
  {
    "nom": "Félicité",
    "age" : 10,
    "race" : "chat de gouttière",
    "aime" : [
      "manger du thon",
      "grimper aux arbres",
      "dormir"
    ],
    "poids": 3.5
  },
  {
    "nom": "Félix",
    "age" : 6,
    "race" : "siamois",
    "aime" : [
      "se lécher",
      "manger des croquettes",
      "dormir"
    ],
    "poids": 3.
  }
]
```

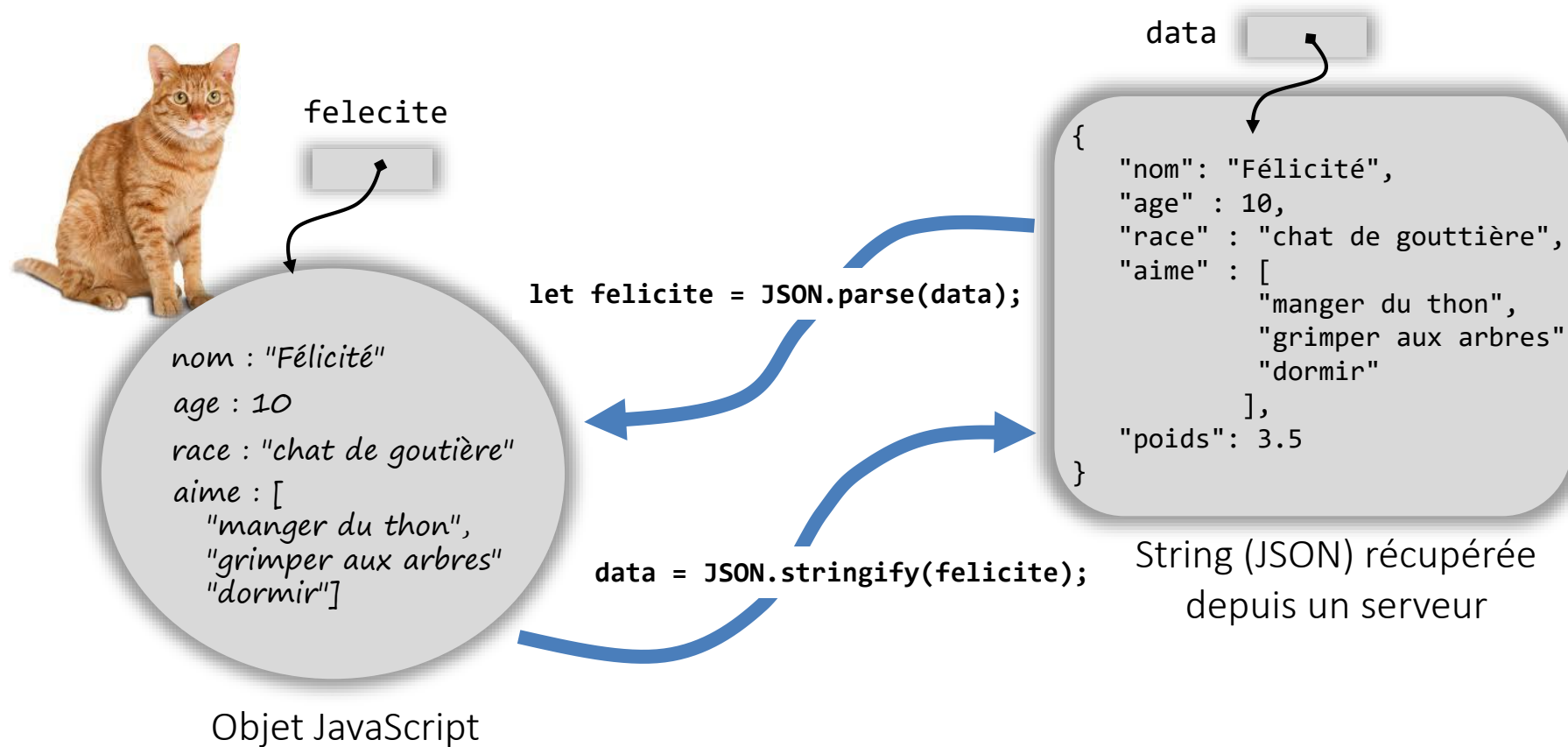

- JSON souvent utilisé comme format commun pour sérialiser et dé-sérialiser des données et les échanger entre des applications qui communiquent sur internet (en particulier via HTTP).



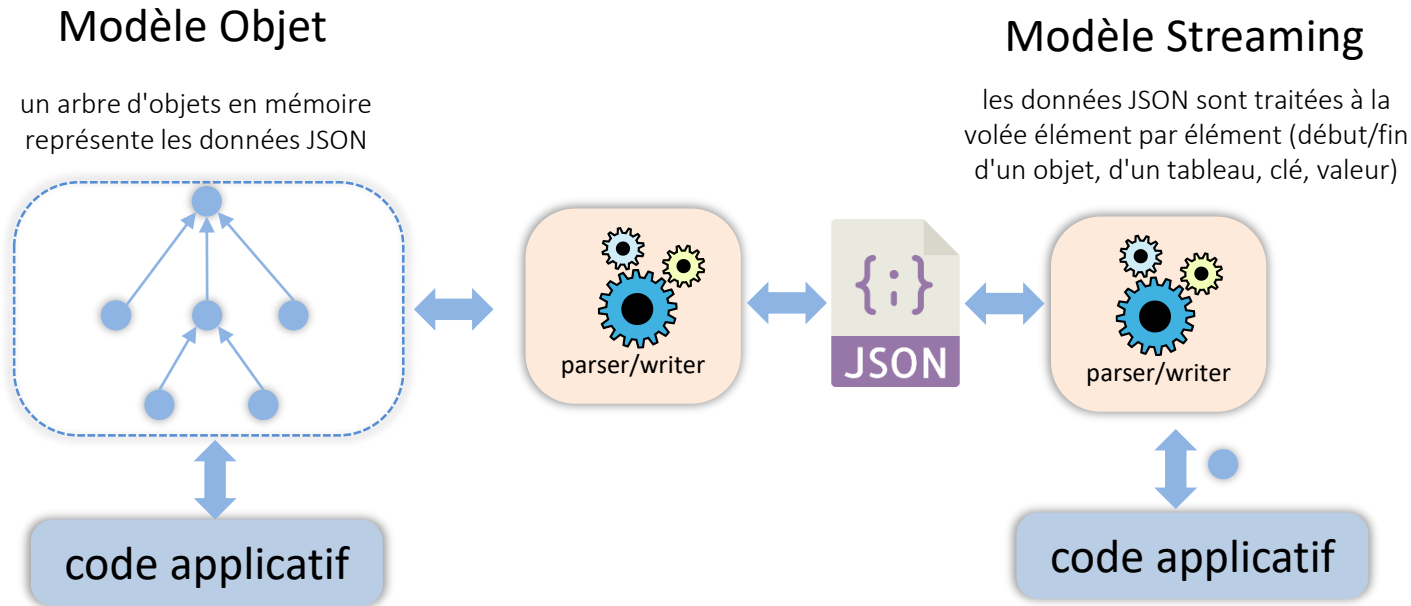
JSON

convertir un objet JavaScript en JSON et inversement

- un usage très courant de JSON est la récupération de données depuis un serveur Web, et d'ensuite les convertir en un objet JavaScript pour les utiliser dans une page web



- deux modèles de programmation pour générer et parser des données JSON



- ✓ flexibilité,
- ✓ possibilité de traitements nécessitant l'accès à l'ensemble de l'arbre
- ✗ occupe plus de mémoire
- ✗ plus lent

- ✓ rapidité
- ✓ faible empreinte mémoire
- ✗ rigidité
- ✗ ne permet que le traitement local d'un élément

JSON

- nombreuses API
 - Gson (<https://github.com/google/gson>)
 - Jackson (<https://github.com/FasterXML/jackson>)
 - JSON Processing (<https://jsonp.java.net/>)
 - ...

The image shows two overlapping screenshots. The background screenshot is the Oracle Java API documentation for the `javax.json` package. The foreground screenshot is the JSON Processing project page on the Oracle website.

JSON Processing
JSR 353: Java API for JSON Processing - Reference Implementation

JSON Processing project is the open source reference implementation of JSR 353 - Java API for JSON query JSON using the streaming API or the object model API.

[Documentation](#) [Download](#)

JSON Processing user guide [JSON Processing is distributed mainly](#)

JSR: Java Specification Requests

intégré à la spécification JEE7

Package javax.json
Provides an object model API to process JSON.
See: Description

Interface Summary

Interface	Description
<code>JsonArray</code>	<code>JsonArray</code> represents an immutable JSON array (an ordered values).
<code>JsonArrayBuilder</code>	A builder for creating <code>JsonArray</code> models from scratch.
<code>JsonBuilderFactory</code>	Factory to create <code>JsonObjectBuilder</code> and <code>JsonArrayBuilder</code> .
<code>JsonNumber</code>	An immutable JSON number value.
<code>JsonObject</code>	<code>JsonObject</code> class represents an immutable JSON object value.

Interfaces

- `JsonArray`
- `JsonArrayBuilder`
- `JsonBuilderFactory`
- `JsonNumber`
- `JsonObject`
- `JsonObjectBuilder`
- `JsonReader`
- `JsonReaderFactory`
- `JsonString`

dépendance Maven

```
<dependency>  
  <groupId>org.glassfish</groupId>  
  <artifactId>javax.json</artifactId>  
  <version>1.1</version>  
</dependency>
```

JSON

JSONProcessing – streaming API javax.json.stream

- générer du JSON

```
FileWriter writer = new FileWriter("test.txt");
JsonGenerator gen = Json.createGenerator(writer);
gen.writeStartObject()
  .write("firstName", "Duke")
  .write("lastName", "Java")
  .write("age", 18)
  .write("streetAddress", "100 Internet Dr")
  .write("city", "JavaTown")
  .write("state", "JA")
  .write("postalCode", "12345")
  .writeStartArray("phoneNumbers")
    .writeStartObject()
      .write("type", "mobile")
      .write("number", "111-111-1111")
    .writeEnd()
    .writeStartObject()
      .write("type", "home")
      .write("number", "222-222-2222")
    .writeEnd()
  .writeEnd()
.gen.writeEnd();
gen.close()
```



```
{
  "firstName": "Duke",
  "lastName": "Java",
  "age": 18,
  "streetAddress": "100 Internet Dr",
  "city": "JavaTown",
  "state": "JA",
  "postalCode": "12345",
  "phoneNumbers": [
    { "Mobile": "111-111-1111" },
    { "Home": "222-222-2222" }
  ]
}
```

JSON

JSONProcessing – streaming API javax.json.stream

- lire du JSON

```
import javax.json.Json;
import javax.json.stream.JsonParser;
...
JsonParser parser = Json.createParser(new StringReader(jsonData));
while (parser.hasNext()) {
    JsonParser.Event event = parser.next();
    switch(event) {
        case START_ARRAY:
        case END_ARRAY:
        case START_OBJECT:
        case END_OBJECT:
        case VALUE_FALSE:
        case VALUE_NULL:
        case VALUE_TRUE:
            System.out.println(event.toString());
            break;
        case KEY_NAME:
            System.out.print(event.toString() + " " +
                parser.getString() + " - ");
            break;
        case VALUE_STRING:
        case VALUE_NUMBER:
            System.out.println(event.toString() + " " +
                parser.getString());
            break;
    }
}
```

objet String
contenant
du JSON

```
{
  "firstName": "Duke",
  "lastName": "Java",
  "age": 18,
  "streetAddress": "100 Internet Dr",
  "city": "JavaTown",
  "state": "JA",
  "postalCode": "12345",
  "phoneNumbers": [
    { "Mobile": "111-111-1111" },
    { "Home": "222-222-2222" }
  ]
}
```

```
START_OBJECT
KEY_NAME firstName - VALUE_STRING Duke
KEY_NAME lastName - VALUE_STRING Java
KEY_NAME age - VALUE_NUMBER 18
KEY_NAME streetAddress - VALUE_STRING 100
Internet Dr
KEY_NAME city - VALUE_STRING JavaTown
KEY_NAME state - VALUE_STRING JA
KEY_NAME postalCode - VALUE_STRING 12345
KEY_NAME phoneNumbers - START_ARRAY
START_OBJECT
KEY_NAME type - VALUE_STRING mobile
KEY_NAME number - VALUE_STRING 111-111-1111
END_OBJECT
START_OBJECT
KEY_NAME type - VALUE_STRING home
KEY_NAME number - VALUE_STRING 222-222-2222
END_OBJECT
END_ARRAY
END_OBJECT
```

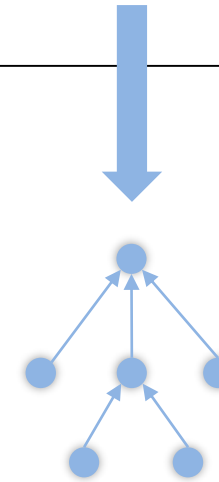
- création d'un modèle objet à partir de données JSON

```
import java.io.FileReader;
import javax.json.Json;
import javax.json.JsonReader;
import javax.json.JsonStructure;
...
JsonReader reader = Json.createReader(
    new FileReader("jsondata.txt"));
JsonStructure jsonst = reader.read()
```

- création d'un modèle objet par un programme Java

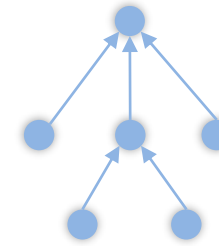
```
import javax.json.Json;
import javax.json.JsonObject;
...
JsonObject model = Json.createObjectBuilder()
    .add("firstName", "Duke")
    .add("lastName", "Java")
    .add("age", 18)
    .add("streetAddress", "100 Internet Dr")
    .add("city", "JavaTown")
    .add("state", "JA")
    .add("postalCode", "12345")
    .add("phoneNumbers", Json.createArrayBuilder()
        .add(Json.createObjectBuilder()
            .add("type", "mobile")
            .add("number", "111-111-1111"))
        .add(Json.createObjectBuilder()
            .add("type", "home")
            .add("number", "222-222-2222")))
    .build();
```

```
{
  "firstName": "Duke",
  "lastName": "Java",
  "age": 18,
  "streetAddress": "100 Internet Dr",
  "city": "JavaTown",
  "state": "JA",
  "postalCode": "12345",
  "phoneNumbers": [
    { "Mobile": "111-111-1111" },
    { "Home": "222-222-2222" }
  ]
}
```



- écrire un modèle objet JSON vers un Stream

```
import java.io.StringWriter;  
import javax.json.JsonWriter;  
...  
StringWriter stWriter = new StringWriter();  
  
JsonWriter jsonWriter = Json.createWriter(stWriter);  
jsonWriter.writeObject(model);  
jsonWriter.close();  
  
String jsonData = stWriter.toString();  
System.out.println(jsonData);
```



```
{  
  "firstName": "Duke",  
  "lastName": "Java",  
  "age": 18,  
  "streetAddress": "100 Internet Dr",  
  "city": "JavaTown",  
  "state": "JA",  
  "postalCode": "12345",  
  "phoneNumbers": [  
    { "Mobile": "111-111-1111" },  
    { "Home": "222-222-2222" }  
  ]  
}
```